SQL

Some methodological considerations to solve complex SQL queries

Types of Queries

Without being exhaustive, we could distinguish the following types:

- **Simple queries with one or more tables, with or without negation of an existential property.**
  - Solution: join the tables, possible use of DISTINCT, no need of subqueries (although can be used).
- **Queries with conditions which depend on aggregated values:**
  - Solution: we need subqueries.
- **Queries with negation of an existential property.**
  - Solution: it is necessary to use subqueries with a NOT IN or a NOT EXISTS.
- **Queries with universal quantification (in the question in natural language we may frequently find the words "all/every" or "only").**
  - Solution: it is necessary to use subqueries with a NOT IN or a NOT EXISTS and, inside the subquery, another negation, which can be a simple non-existential negation (<>) or an existential negation (NOT EXISTS or NOT IN).
- **Queries with direct and aggregated values in the SELECT or which can be expressed as "for each X list …".**
  - Solution: a GROUP BY is needed. We have to group by the X identifier and by the attributes which appear in the SELECT clause.
- **Queries where we want that when we join two tables, we get the elements of one table which do not have correspondence with the elements of the other table.**
  - Solution: it is necessary to use LEFT/RIGHT JOIN or the UNION operator.
- **Combinations of the previous types.**

SIMPLE QUERIES OVER SEVERAL TABLES, WITH OR WITHOUT NEGATION OF ONE NON-EXISTENTIAL PROPERTY.

Solution: join the tables, maybe DISTINCT, no need of subqueries (although can be used).

Example: List the number of the stages ('etapas') won by racers ('ciclistas') which belong to teams ('equipos') such that its coach ('director') has a name which begins with 'A'.

Solution with subqueries *(in this case we don't need DISTINCT, because the stages have only one winner, although can be added without affecting the result).*

```
SELECT DISTINCT E.netapa FROM Etapa E, Ciclista C, Equipo Q
WHERE E.dorsal=C.dorsal AND C.nomeq = Q.nomeq AND E.director LIKE 'A%';
```

Solution with subqueries (DIVIDE AND CONQUER). First (from right to left) we get the names of the teams with coaches whose name begins with 'A', then the racer numbers ('dorsal') of the racers from these teams and then the stages won by these racers.

```
SELECT netapa FROM Etapa
WHERE dorsal IN (SELECT dorsal FROM Ciclista
                 WHERE nomeq IN (SELECT nomeq FROM Equipo
                                 WHERE director LIKE 'A%'));
```

Example: List the names of the racers which are **NOT** older than 18 years old..

The question contains a negation, but it only affects a condition over one attribute.

```
SELECT C.nombre FROM Ciclista C WHERE NOT (C.edad >= 18);
```

We can also put C.edad < 18, more simply.

QUERIES WITH CONDITIONS WHICH DEPEND ON AGGREGATED VALUES:

Solution: we need subqueries.

Example: List the name of the mountain passes ('puertos') with an altitude which is greater than the **mean** of the altitude of all the mountain passes of category 2.

```
SELECT nompuerto FROM Puerto
WHERE altura > (SELECT AVG(altura) FROM Puerto WHERE categoría = 2 );
```

Example: List the name of the mountain passes and the racers ('ciclistas') which have won them, such that the pass has the **greatest** slope ('pendiente').

Solution with a MAX (aggregated function).

```
SELECT P.nompuerto, C.nombre FROM Puerto P, Ciclista C
WHERE P.dorsal = C.dorsal  AND
P.pendiente = (SELECT MAX(P1.pendiente) FROM Puerto P1 );
```

Solution with a >= ALL.

```
SELECT P.nompuerto, C.nombre FROM Puerto P, Ciclista C
WHERE P.dorsal = C.dorsal  AND
P.pendiente >= ALL (SELECT P1.pendiente FROM Puerto P1 );
```

QUERIES WITH NEGATION OF AN EXISTENTIAL PROPERTY.

Solution: we need to use subqueries with a NOT IN or a NOT EXISTS.

Example: List el name de the racer which has **NOT** won any stage.
The negation doesn't affect a simple attribute, but affects the existence or not of row in the "etapa" table for that racer. In this case, then, we have the negation of an existential property (to win stages), i.e., the property is "that there exists a stage which has been won by the racer". Its negation is "that there does not exist a stage which has been won by the racer". In SQL…

Solution with subquery with NOT EXISTS.

```
SELECT C.nombre FROM Ciclista C
WHERE NOT EXISTS (SELECT * FROM Etapa E
                  WHERE E.dorsal = C.dorsal);
```

Solution with subquery with NOT IN.

```
SELECT name FROM Ciclista
WHERE dorsal NOT IN (SELECT dorsal FROM Etapa);
```

But the following one is COMPLETELY WRONG:

```
SELECT C.nombre FROM Ciclista C, Etapa E
        WHERE E.dorsal <> C.dorsal;
```

QUERIES WITH UNIVERSAL QUANTIFICATION

(in the question in natural language we may frequently find the words "all/every" or "only").

Solution: It's necessary to use subqueries with a NOT IN or a NOT EXISTS and, inside the subquery, another negation, which can be a simple non-existential negation (<>) or an existential negation (NOT EXISTS o NOT IN).

Example: List the name of the racer (if there is one) which has won **all** the stages with more than 200 km.

Solution: we transform the query to "undo" the word "all", with the following schema:

$\forall x\ p \quad \Rightarrow \quad \neg\exists x \neg p$ NOT EXISTS (…2nd negation….)

This converts the query into "List the name of the racer such that there does ***not*** exist a stage with more than 200 km. which he has ***not*** won" which we can now express in SQL.

```
SELECT name FROM Ciclista C
WHERE NOT EXISTS (SELECT * FROM Etapa E
                  WHERE km > 200 AND C.dorsal <> E.dorsal );
```

It is very important to know which part is negated. Here, the condition "km > 200" must remain unaltered, but the condition "C.dorsal = E.dorsal" must be negated, because we want to say that he has ***not*** won the stage.

QUERIES WITH UNIVERSAL QUANTIFICATION (ADD-ONS)

"ADD-ONs" in queries with universal quantification:

SAME example as before: List the name of the racer (if there is one) which has won **all** the stages with more than 200 km.

```
SELECT C.nombre FROM Ciclista C
WHERE NOT EXISTS (SELECT * FROM Etapa E
                  WHERE E.km > 200 AND C.dorsal <> E.dorsal );
```

What happens if there are no stages with more than 200 km?

WE WOULD LIST ALL THE RACERS!!!

Solution:

```
SELECT C.nombre FROM Ciclista C
  WHERE NOT EXISTS (SELECT * FROM Etapa E
                    WHERE E.km > 200 AND C.dorsal <> E.dorsal )
        AND EXISTS (SELECT * FROM ETAPA E2 WHERE E2.km > 200);
```

QUERIES WITH UNIVERSAL QUANTIFICATION

Example with two not exists: “List the name of the racers which have worn all the jerseys (maillots)”.
Solution: we transform it into “List the name of the racers such that there is no jersey which hasn’t been worn by them”. Or in order to see it clearer with two negations of the two existential quantification, we can express it as:

“List the name of the racers such that there does not exist a jersey for which there does not exist a row in “llevar” which connects the racer and the ciclist”

```
SELECT DISTINCT C.nombre FROM ciclista C
WHERE NOT EXISTS (SELECT * FROM maillot M
                  WHERE NOT EXISTS (SELECT * FROM llevar L
                                    WHERE M.codigo=L.codigo AND
                                          C.dorsal = L.dorsal));
```

QUERIES WITH UNIVERSAL QUANTIFICATION

Example with "ONLY": List the colour of the jerseys which have **only** been worn by racers of the same team.

Solution: these queries have the same schema as the universal quantification; generally, these queries can be translated into expressions like "that there is one and there is no other or different".

In this case, we transform the previous query into "List the colour of the jerseys which have been worn by a racer of **one** team and which have not been worn by a racer of a **different** team"

```
SELECT DISTINCT color FROM maillot m, llevar l, ciclista c
WHERE c.dorsal=l.dorsal AND m.codigo=l.codigo
        AND NOT EXISTS (SELECT * FROM llevar l2, ciclista c2
                        WHERE c2.dorsal=l2.dorsal AND
                        c2.nomeq<>c.nomeq AND l2.codigo=l.codigo);
```

QUERIES WITH UNIVERSAL QUANTIFICATION

NEW QUANTIFIERS IN SQL3 (SQL99): FOR ALL and FOR SOME

"List the name of the racers which have worn all the jerseys (maillots)".
Even though we have the "FOR ALL" operator we need some transformation

Solution: we transform it into "List the name of the racers such that for every jersey it has been worn by them".

```
SELECT DISTINCT C.nombre FROM ciclista C
WHERE FOR ALL (SELECT * FROM maillot M)
                    (EXISTS (SELECT * FROM llevar L
                                WHERE M.codigo=L.codigo AND
                                      C.dorsal = L.dorsal));
```

QUERIES WITH DIRECT AND AGGREGATED VALUES IN THE SELECT OR WHICH CAN BE EXPRESSED AS "FOR EACH X LIST …".

Solution: a GROUP BY is needed. We have to group by the X identifier and by the attributes which appear in the SELECT clause.

Example: List the name of **each** team and the average age of the racers of the team:

```
SELECT nomeq, AVG(edad) FROM Ciclista
GROUP BY nomeq;
```

Use of WHERE: conditions over the individuals (the rows).
Use of HAVING: conditions over the groups and only using the attributes which appear in the GROUP BY.

Example: List the name of the racers and the number of mountain passes their have won, being their average slope greater than 10.

```
SELECT C.nombre, COUNT(P.nompuerto)
FROM Ciclista C, Puerto P
WHERE C.dorsal = P.dorsal
GROUP BY C.dorsal, C.nombre      /* We use PK to group*/
HAVING AVG (P.pendiente) >10;
```

JOIN QUERIES SUCH THAT WE WANT TO GET THE ELEMENTS OF ONE TABLE WHICH DO NOT HAVE CORRESPONDENCE WITH THE ELEMENTS OF THE OTHER TABLE.

Solution: it is necessary to use LEFT/RIGHT JOIN or the UNION operator.

Example: List **for each team** the total of racers in the team. The teams without racers mustn't appear.

```
SELECT nomeq, COUNT(*) FROM Ciclista GROUP BY nomeq;
```

A query which is equivalent to the previous one:

```
SELECT E.nomeq, COUNT(*) FROM Ciclista C, Equipo E
WHERE C.nomeq = E.nomeq GROUP BY nomeq;
```

Example: List **for every team** the total of racers in the team. The teams without racers MUST appear.

```
SELECT E.nomeq, COUNT(C.dorsal)
FROM Ciclista C RIGHT JOIN Equipo E ON E.nomeq = C.nomeq
GROUP BY nomeq;
```

Now we would have the teams without racers, and with a count equal to 0. It is important to count (C.dorsal), because the row which is generated for the team without racers is filled up with nulls, and COUNT(*) would count them (we would have then that the teams without racers would seem to have one racer).

(The previous query can also be solved with a UNION).

MORE EXAMPLES FOR THE 'CICLISMO' SCHEMA

1) List the stage number and the start city of those stages which have no mountain passes.

- *Consulta el número de las etapas y la ciudad de salida de aquellas etapas que no tengan puertos de montaña.*

```
SELECT netapa, salida
FROM etapa
WHERE  not exists ( SELECT * FROM puerto
                    WHERE puerto.netapa=etapa.netapa );
```

MORE EXAMPLES FOR THE 'CICLISMO' SCHEMA

2) List the name of the start and arrival cities of the stage with the mountain pass with greatest slope.

- *Consulta el nombre de la ciudad de salida y de llegada de la etapa donde está el puerto con mayor pendiente.*

```
SELECT e.salida, e.llegada
FROM etapa e, puerto p
WHERE e.netapa=p.netapa AND
    p.pendiente=(select MAX(pendiente) from puerto );
```

MORE EXAMPLES FOR THE ‘CICLISMO’ SCHEMA

3) Who is the youngest racer?

- *¿Quién es el ciclista más joven?* .

```
SELECT name
FROM ciclista c
WHERE edad = ( SELECT MIN(edad) FROM ciclista );
```

4) List the name of the racers which have won all the mountain passes of one stage and which have also won that stage. (Exercise 17)

- *Consulta el nombre de los ciclistas que han ganado todos los puertos de una etapa y además han ganado esa misma etapa. (Ej. 17 boletín)*

```
SELECT c.nombre FROM ciclista c, etapa e
WHERE e.dorsal=c.dorsal AND
NOT EXISTS( SELECT * FROM puerto p
                    WHERE p.netapa=e.netapa
                    AND c.dorsal <> p.dorsal )
AND EXISTS ( SELECT * FROM puerto p
                    WHERE p.dorsal=c.dorsal
                    AND p.netapa=e.netapa );
```

5) List the colour of those jerseys which have only been worn by racers of a single team.

•*Consulta el color de aquellos maillots que sólo han sido llevados por ciclistas de un mismo equipo.*

```
SELECT DISTINCT color FROM maillot m, llevar l, ciclista c
WHERE c.dorsal=l.dorsal AND m.codigo=l.codigo
 AND NOT EXISTS( SELECT * FROM llevar l2, ciclista c2
                          WHERE c2.dorsal=l2.dorsal AND
                           c2.nomeq<>c.nomeq AND l2.codigo=l.codigo);
```

6) List the name of the racers which belong to a team which has more than 5 racers, also showing the number of stages won by each.

•*Consulta el nombre de los ciclistas que pertenezcan a un equipo que tenga más de cinco corredores indicando el número de etapas ganadas por cada uno.*

```
SELECT c.nombre, COUNT(*) FROM ciclista c, etapa e
WHERE c.dorsal=e.dorsal AND
                5<( SELECT COUNT(*) FROM ciclista c2
                    WHERE c2.nomeq=c.nomeq )
GROUP BY c.nombre, c.dorsal;
```

7) List the name of the teams which have the maximum average age of all the teams.

•*Consulta el nombre de los equipos que tengan la media de edad máxima de todos los equipos.*

```
SELECT C.nomeq, AVG(C.edad)
FROM ciclista C
GROUP BY C.nomeq
HAVING AVG(C.edad) >= ALL
                        (SELECT AVG(D.edad)
                        FROM ciclista D
                       GROUP BY D.nomeq);
```

8) List the name of the racers which have not worn all the jerseys which have been worn by the racer with number (dorsal) 1.

•*Consulta el nombre de los ciclistas que no han llevado todos los maillots que ha llevado el ciclista de dorsal 1.*

```
SELECT c.nombre FROM ciclista c
WHERE EXISTS( SELECT * FROM llevar l
                    WHERE l.dorsal=1 AND
          NOT EXISTS(SELECT * FROM llevar l2
                    WHERE l2.dorsal=c.dorsal AND
                    l2.codigo=l.codigo) );
```